**ANL252**

**Python for Data Analytics**

**End-of-Course Assessment – July Semester 2021**

|  |  |
| --- | --- |
| Full Name | Cheong Jia En, Sandras |
| PI No. | N1981099 |

Submission Date: 12 September 2021

**Question 1a**

(i)

**Code**

import pandas as pd

# Reading in original csv file

# Missing values "." are indicated as NaN

ship = pd.read\_csv("ship.csv", na\_values = ".")

ship

**Output**

Table

Description automatically generated with medium confidence

Table

Description automatically generated

(ii)

**Code**

# Renaming columns

ship = ship.rename(columns = {"T": "types", "A": "c\_years", "P": "o\_periods", "MS": "s\_months", "Y": "incidents"})

ship.head()

**Output**

Table

Description automatically generated

(iii)

**Code**

# First group based on "types"

# Within each type, group based on "o\_periods"

shipgroup = ship.groupby(["types", "o\_periods"])

# Compute mean of s\_months and incidents based on the grouping

# Round the mean values

shipgroup = shipgroup[["s\_months", "incidents"]].mean().round()

# Set the mean values as nearest integers

shipgroup = shipgroup.astype({"s\_months": "int", "incidents": "int"})

display(shipgroup)

**Output**

Table

Description automatically generated with low confidence

(iv)

**Code**

# Locate all the missing values by rows

missing = ship.isnull().any(axis = 1)

ship.loc[missing[missing == True].index]

**Output**

Table

Description automatically generated

**Code**

# Left-hand side: Using iloc to select the row with NaN from ship dataframe

# Right-hand side: Using fillna to fill NaN values in the specified row

# using loc to select the computed mean of s\_months and incidents from shipgroup dataframe

# from the row with corresponding types and o\_periods

ship.iloc[6] = ship.iloc[6].fillna(shipgroup.loc[(1,1)])

ship.iloc[14] = ship.iloc[14].fillna(shipgroup.loc[(2,1)])

ship.iloc[22] = ship.iloc[22].fillna(shipgroup.loc[(3,1)])

ship.iloc[30] = ship.iloc[30].fillna(shipgroup.loc[(4,1)])

ship.iloc[33] = ship.iloc[33].fillna(shipgroup.loc[(5,2)])

ship.iloc[38] = ship.iloc[38].fillna(shipgroup.loc[(5,1)])

ship = ship.astype({"s\_months": "int", "incidents": "int"})

ship

**Output**

Table

Description automatically generated

Table

Description automatically generated

(v)

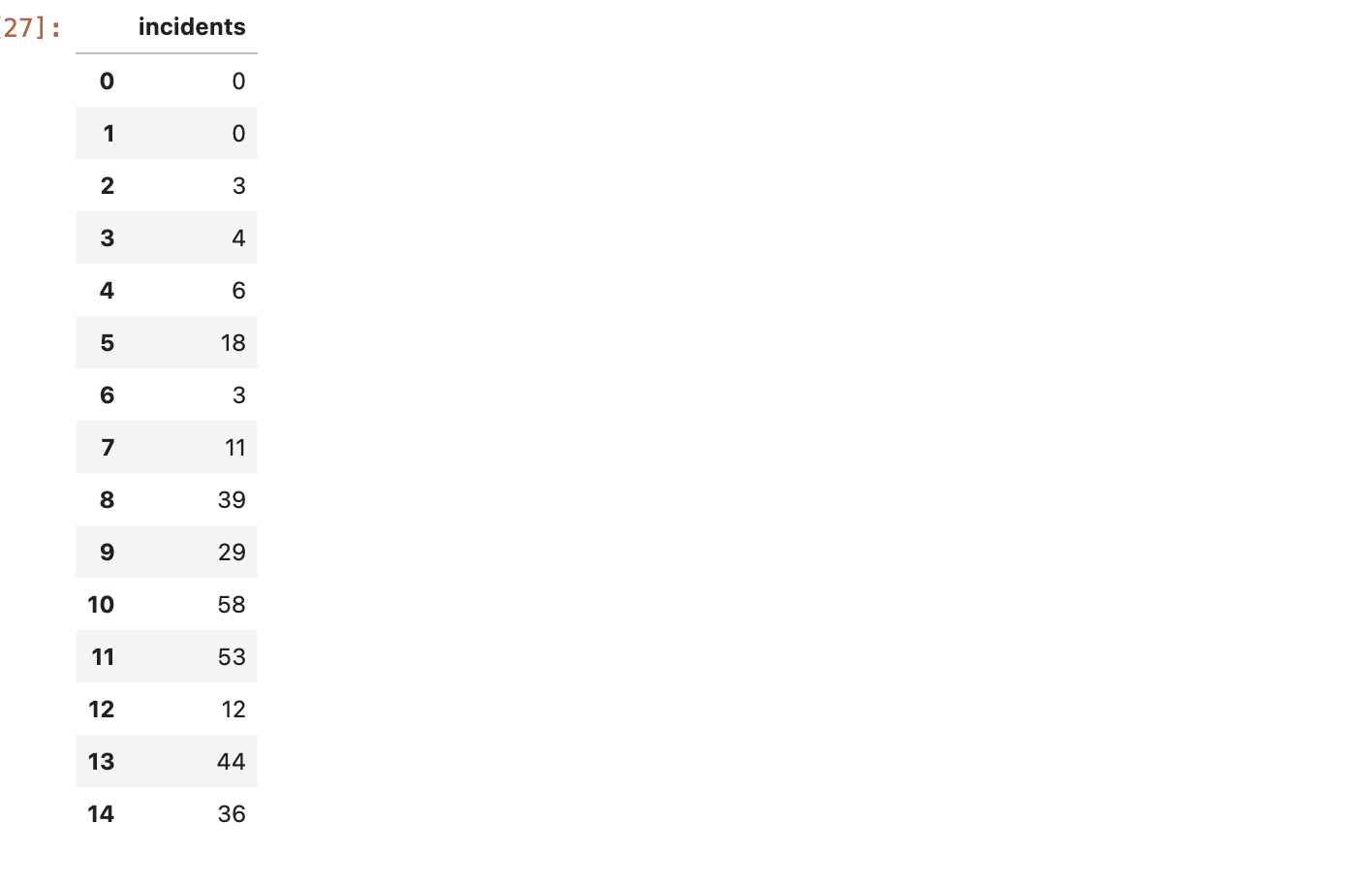
**Code**

# Save target variable 'incidents' in a pandas DataFrame named "Y"

Y = pd.DataFrame(ship, columns = ['incidents']).astype({"incidents": "int"})

Y.head(15)

**Output**



**Question 1b**

(i)

**Code**

# Converting the 3 variables to categorical

ship = ship.astype({'types': 'category', 'c\_years': 'category', 'o\_periods': 'category'})

# Checking the data types of the variables

ship.dtypes

**Output**



(ii)

**Code**

# Creating dummy variables from categorical variables

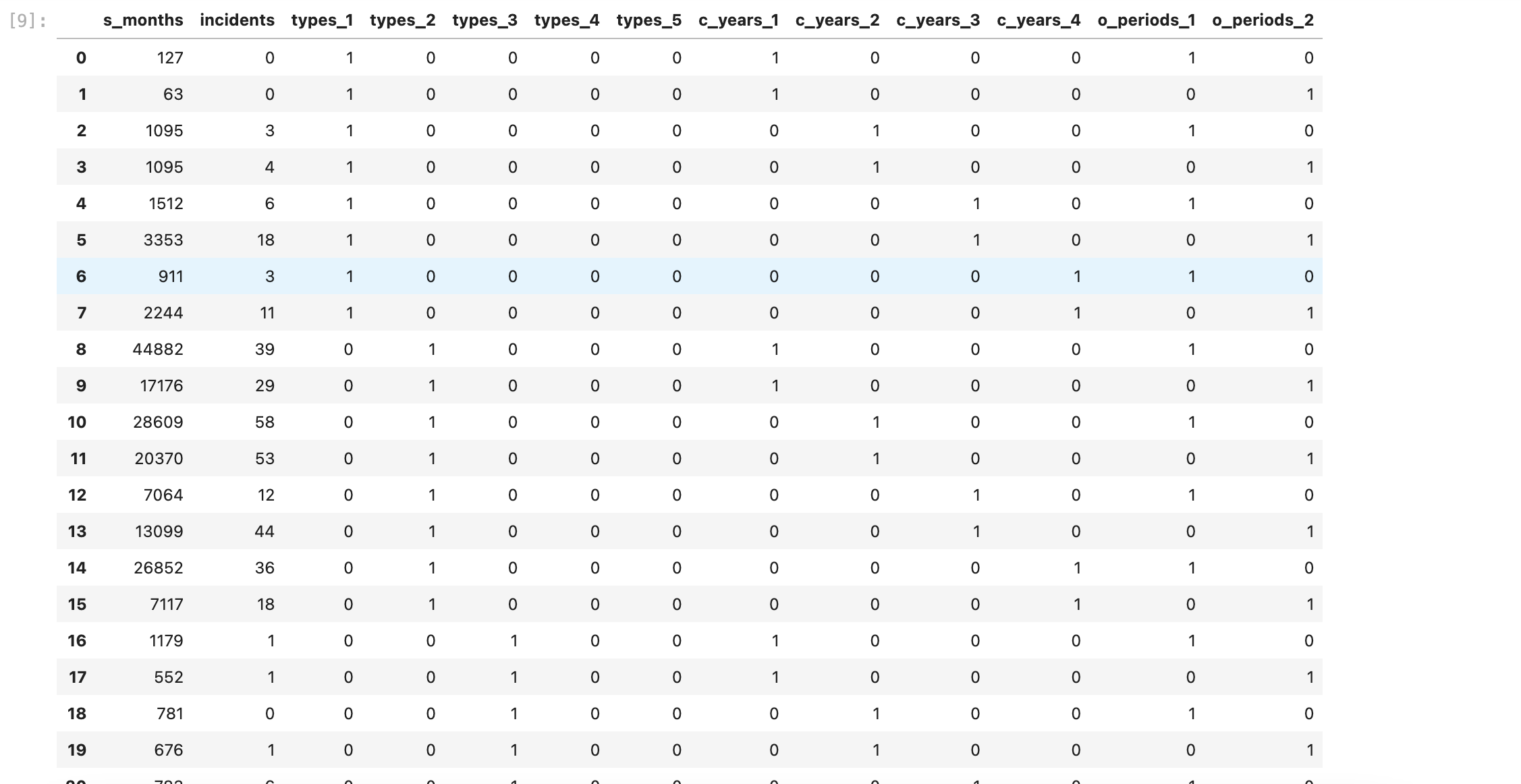
shipdummy = pd.get\_dummies(ship)

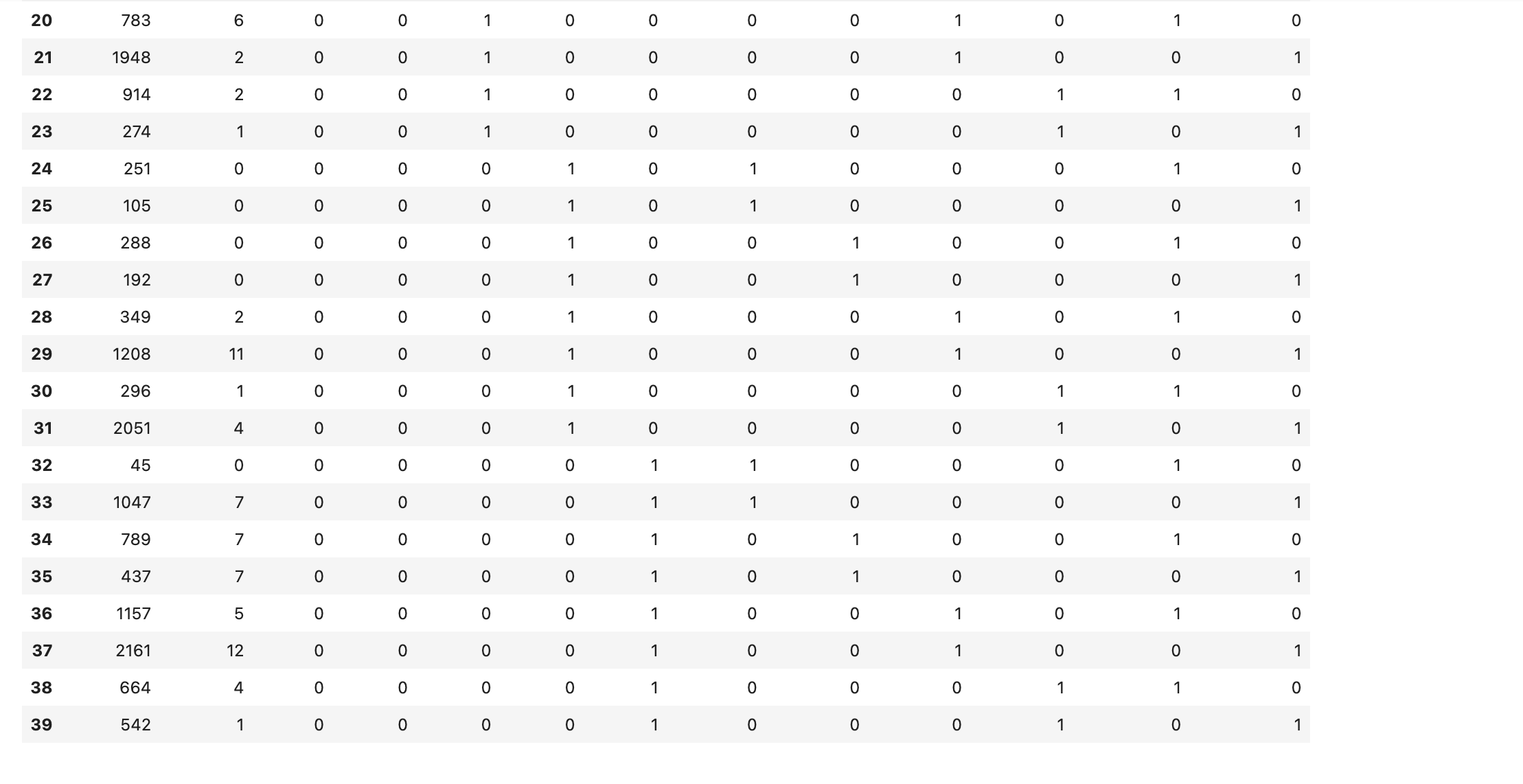
# Save results as a pandas DataFrame named "X"

X = pd.DataFrame(shipdummy)

X

**Output**





(iii)

**Code**

import numpy as np

# Perform log transformation on the numeric variable 's\_months'

# Save the transformed variable in both DataFrames "ship" and "X"

# Did not round to have more precise and accurate values

ship["log\_s\_months"] = np.log(ship["s\_months"])

X["log\_s\_months"] = np.log(X["s\_months"])

# Print DataFrame "X" where transformed variable is attached

X

**Output**

Application, table

Description automatically generated

Table

Description automatically generated

**Code**

# Print DataFrame "ship" where transformed variable is attached

ship.head(10)

**Output**

Table

Description automatically generated

**Question 1c**

The dataset consists a total of only 40 records, which is too small to accurately evaluate the model’s predictive performance. When this dataset is split into training and testing sets, there will be insufficient data in the training dataset for the model to learn an effective mapping of inputs relative to outputs (Brownlee, 2020). Likewise, the testing set would be an inaccurate representation of the original dataset to achieve an effective evaluation of model performance (Brownlee, 2020). It may further lead to an overly optimistic or overly pessimistic estimated performance (Brownlee, 2020). This is also known as overfitting or underfitting. Therefore, it would be more suitable to use the entire dataset for training purpose to ensure good performance and good generalisation of the data.

**Question 1d**

**Code**

import sqlite3

# Save DataFrame 'ship' to new csv file without index

ship.to\_csv('ship\_prepared.csv', index = False)

# Establish a Python connection to a new database called "ship.db" and

# create a cursor object for later use

conn = sqlite3.connect("ship.db")

cursor = conn.cursor()

# Import newly saved csv text file

newship = pd.read\_csv("ship\_prepared.csv")

# Export pandas DataFrame to Database by SQL

newship.to\_sql("newship", conn, if\_exists = "replace", index = False)

# Select table from "ship.db" database

cursor.execute("SELECT \* FROM newship;")

# Fetch all records to verify

print(cursor.fetchall())

**Output**

Text

Description automatically generated

**Question 2a**

The corresponding scikit-learn module is “linear\_model”. It is a class of the scikit-learn package i.e., sklearn, where the term Linear Model itself implies that the model describes a dependent variable in terms of a linear combination of independent variables. It comprises of several estimators. Some examples are “LinearRegression” and “LogisticRegression”.

The relevant estimator used in this question is “PoissonRegressor” as Poisson Regression is used for this case study. This suggests that the target variable Y is assumed to have a Poisson distribution and it is usually suitable for target variable with small integers. This estimator has several parameters:

1. “alpha” which is a constant,
2. “fit\_intercept” that determines if an intercept should be included in the linear prediction,
3. “max\_iter” which is the maximum number of iterations,
4. “tol” which is the stopping criterion,
5. “verbose” which is the verbosity and
6. “warm\_start” when set to true, the previously fitted model attributes such as coefficient and intercept will be used to initialise the new model in subsequent fittings.

The fit function is included in every estimator and it takes the target data Y and input data X as arguments to fit a generalised linear model. The shape of arrays X and Y have to be the same, else an error “ValueError” will be raised. The fit function takes training data to fit the model whereas the predict function takes testing data such as the sample X to perform predictions.

**Question 2b**

**Code**

# Import required packages

import pandas as pd

import numpy as np

import sklearn

# Import model functions from scikit-learn

from sklearn import linear\_model

# Converting Y dataframe into a 1D array

Y = np.ravel(Y)

print(f"DataFrame Y : ")

Y

**Output**

A picture containing circle

Description automatically generated

**Code**

# Remove column "s\_months" and "incidents" from dataframe X so only

# independent variables remain

X.drop(columns = ['s\_months', 'incidents'], inplace = True)

print(f"\nDataFrame X: \n")

X

**Output**

**Table

Description automatically generatedTable

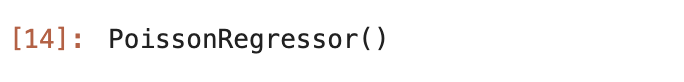
Description automatically generated**

**Code**

pr = linear\_model.PoissonRegressor()

pr.fit(X, Y)

**Output**

****

**Code**

# D^2 = Percentage of deviance explained

score = pr.score(X, Y)

score

**Output**

****

**Code**

# Coefficients of Beta 1 to 12

coef = pr.coef\_

beta1to12 = pd.DataFrame(coef, columns=['Coefficients'], index=['Beta1','Beta2','Beta3','Beta4','Beta5','Beta6',

'Beta7','Beta8','Beta9','Beta10','Beta11','Beta12',])

beta1to12

**Output**

**Table

Description automatically generated with low confidence**

**Code**

# Obtain coefficient of Beta 0

intercept = pr.intercept\_

beta0 = pd.DataFrame(intercept, columns=['Coefficients'], index=['Beta0'])

beta0.append(beta1to12)

**Output**

Table

Description automatically generated

**Code**

# Parameters of this estimated model

pr.get\_params(deep=True)

**Output**

**Text

Description automatically generated with low confidence**

**Question 2c**

**Code**

# Make Y into a list

Ylist = Y.tolist()

# Find all predicted Y values i.e. E(Y)

EY = pr.predict(X)

EY\_int = EY.astype(int)

EYlist = EY\_int.tolist()

print(f"Actual Y values: \n{Ylist}")

print(f"\nE(Y) values: \n{EY}")

**Output**

Graphical user interface, text, table

Description automatically generated with medium confidence

**Code**

# Note:

# Y = 0 for Index 0, 1, 18, 24, 25, 26, 27, 32

# Inserting the formula

d = Ylist/EY

dlog = np.log(d)

# Converting negative infinity values to 0

dlog[dlog == -np.inf] = 0

D = 2\*(sum((Ylist\*dlog)-(Ylist-EY)))

D

**Output**

****

**References**

Brownlee, J. (2020, July 24). *Train-Test Split for Evaluating Machine Learning Algorithms*. Retrieved from <https://machinelearningmastery.com/train-test-split-for-evaluating-machine-learning-algorithms/>